uc3m | Universidad Carlos III de Madrid

Grado en Ingeniería Electrónica Industrial y Automática

Curso 2018-2019

Trabajo Fin de Grado

Desarrollo de Aplicación Android para comunicación DLMS/COSEM con Smart Meter

---

David Villanueva Arteaga

Tutor

Juan Miguel Gómez Berbís

Leganés, Madrid, junio de 2019

RESUMEN

La presente memoria corresponde al estudio que se ha llevado a cabo sobre las smart grids (en castellano redes eléctricas inteligentes) y, en concreto, sobre los smart meters (en castellano contadores inteligentes) como componente indispensable de la smart grid. El estudio se ha enfocado en las ventajas que presenta la smart grid respecto a la red eléctrica convencional.

Por otra parte, se estudia el protocolo de comunicación DLMS/COSEM, y su versión IEC 62056 que se utiliza para establecer conexión e intercambiar datos con smart meters.

Juntando ambos conceptos se ha logrado desarrollar una aplicación Android que permite la conexión de un smartphone con un smart meter para poder extraer toda la información que tiene disponible, y/o modificar sus parámetros de configuración, quedando reflejada la documentación de la misma en esta memoria.

La principal funcionalidad de la aplicación es que la automatización de la lectura de un smart meter, donde toda la información recopilada es subida a un servidor de forma instantánea.

Palabras claves: Smart grid, Smart meter, DLMS/COSEM, Smartphone, Android.

ÍNDICE DE CONTENIDO

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

1.1. Motivación

1) Automatización y programación

Vivimos en un mundo en el que la automatización del mismo avanza sin parar en todos los ámbitos de nuestra sociedad, haciéndonos la vida cada vez más fácil.

Toda máquina que realiza un proceso automatizado necesita una tarea de programación que le explique a través de un lenguaje, las funciones que debe realizar para desarrollar una tarea o proceso.

Por este motivo, uno de los principales requisitos de la automatización es la programación informática, que consiste en realizar tareas repetitivas en un ordenador mediante un conjunto de métodos.

El auge de la programación se produjo en los años 80 donde aparece el lenguaje C++, y posteriormente con la llegada de internet ha sufrido un crecimiento exponencial, del cual no sabemos cuáles serán sus límites.

Aunque la programación se empezó a desarrollar en ordenadores, este ha dado paso a los smartphones, que han sufrido una revolución en la última década hasta tal punto de que, actualmente, en el mundo existen más dispositivos móviles que personas habitando nuestro planeta. [1]

Por este motivo, aparece la necesidad del desarrollo de aplicaciones móviles que faciliten la vida de las personas, ya sea ofreciendo entretenimiento o ayudando a realizar con mayor facilidad las tareas del día.

2) Red eléctrica en España

La infraestructura de la red eléctrica tradicional que se tiene instalada en la mayoría de las casas españolas sigue siendo la misma desde que en 1944 se crearon la red eléctrica primaria a nivel nacional, que interconectaba los distintos sistemas eléctricos regionales.

En la red eléctrica convencional, la energía eléctrica circula desde las centrales de producción hasta las zonas de consumo a través de las líneas de alta tensión, subestaciones, transformadores y demás infraestructuras, tal como se puede observar en la *ilustración 1.1.*

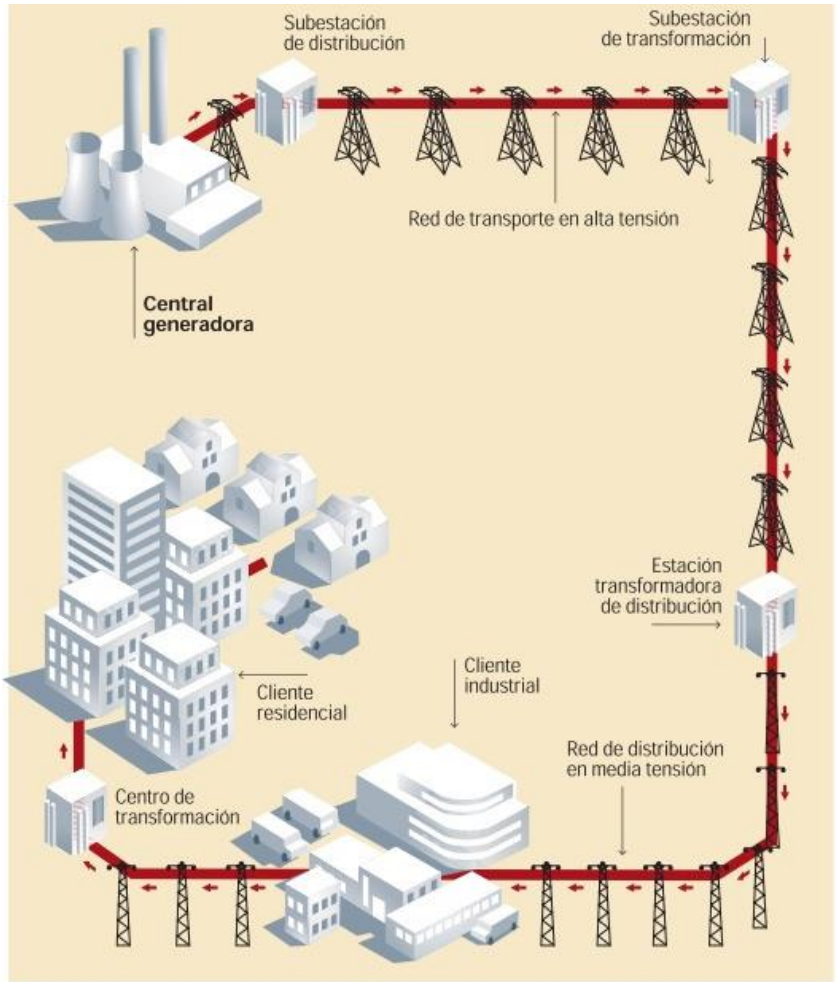

Ilustración 1.1. Esquema de una red eléctrica convencional [2]

Desde sus inicios, la estructura del sistema de la eléctrica tiene una distribución geográfica donde las centrales de generación de energía están muy alejadas de los principales centros de consumo. “Esta disposición implica grandes pérdidas por transporte y distribución de la energía, al tener que cubrir grandes distancias. Cada vez más, se hace necesaria una reestructuración que convierta estas redes en redes más efectivas y robustas de forma que puedan soportar las necesidades futuras, tanto desde el punto de vista del consumidor como del medioambiente”. [3]

La evolución de la red eléctrica ha llegado de la mano de la **smart grid**, y aunque en España aún hay pocas en funcionamiento, somos uno de los países pioneros en el despliegue de **smart meters** [4], siendo estos un elemento clave de la **smart grid**.

Con las **smart grids** llega una nueva revolución de la digitalización, aplicada a la vida diaria de las personas. La posibilidad de poder controlar todos y cada uno de los aparatos de casa desde un teléfono móvil sin necesidad de que el usuario se encuentre en ella o controlar en todo momento el gasto de energía eléctrica, es el presente en el que nos encontramos.

Por los dos conceptos planteados, programación y red eléctrica, la **motivación** de este trabajo de fin de grado es juntar ambos aspectos, que se encuentran en pleno auge para desarrollar una aplicación para dispositivos Android que permita establecer conexión con un smart meter y automatizar el proceso de lectura de este.

1.2. Objetivo

El objetivo de este TFG consiste en el desarrollo de una aplicación para dispositivos **Android** que permita la conexión con un **smart meter**, para leer la información almacenada en su interior.

La principal funcionalidad que debe realizar la aplicación es permitir que la **lectura** se realice **automáticamente**, donde el usuario que interactúe con la aplicación solo tenga que pulsar un botón y el resto del proceso de ejecute por sí solo. Finalmente, cuando acabe la lectura, toda la información se subirá automáticamente a un servidor, disponiendo así de la información en tiempo real.

La conexión física entre el smartphone y el smart meter se realiza con el uso de una sonda óptica, mediante el protocolo DLMS/COSEM.

Todos los componentes de las smart grids, incluido el smart meter, tienen acceso y control remoto desde una zona centralizada donde una compañía puede gestionarlos. Sin embargo, al ser una tecnología emergente y en estado de evolución pueden existir problemas de comunicación puntuales que impidan un acceso momentáneo. Por este motivo, es necesario disponer de una herramienta que respalde este tipo de incidencias y permita realizar lecturas directas como solución de campo, que suplan los fallos de lectura remota. La aplicación que se desarrolla en esta memoria proporciona este soporte.

Los datos escogidos para lectura han sido los que se han creído más útiles, partiendo de la base de que el smart meter con el que se ha trabajado no está conectado a ninguna instalación eléctrica, por lo que el registro de información que tenía era mucho menor del potencial.

1.3. Marco Regulador

En España existen leyes que regulan la implantación de smart meters, sin embargo, aún no existen normativas que regulen el uso conjunto de la integración de estos en smart grids.

Las leyes y los reales decretos que marcan la regulación actual de los smart meters son las siguientes:

- Real Decreto 1110/2007, de 24 de agosto.
  "Se aprueba el Reglamento unificado de puntos de medida del sistema eléctrico y se exige, de modo necesario, la implantación de un sistema de medidas homogéneo y efectivo de los tránsitos de energía entre las diversas actividades eléctricas." [5]

- Orden ITC/3860/2007, de 28 de diciembre.
  Establece un plan de sustitución a plazos de contadores eléctricos: "Todos los contadores de medida en suministros de energía eléctrica con una potencia contratada de hasta 15 kW deberán ser sustituidos por nuevos equipos que permitan la discriminación horaria y la telegestión antes del 31 de diciembre de 2018." [6]

2. ESTADO DEL ARTE

2.1. Smart grid

Smart grid (en castellano red eléctrica inteligente) se define como “una red eléctrica capaz de integrar de forma inteligente el comportamiento y las acciones de los usuarios conectados a ella (quienes generan electricidad, quienes consumen y quienes realizan ambas acciones) para proporcionar un suministro de electricidad seguro, económico y sostenible”. [7]

Gestionan de forma eficiente la energía eléctrica mediante el uso de equipos de control, monitorización, y auto diagnóstico que trabajan conjuntamente para agrupar todas las áreas de la red en un solo sistema de gestión, ofreciendo así un uso racional y eficiente de la electricidad.

La principal característica que hace “inteligente” a una smart grid es la incorporación de la tecnología necesaria para poder establecer, además de un flujo de energía eléctrica, un fujo de datos, convirtiéndose así en una **red de comunicación bidireccional** entre el usuario y la instalación.

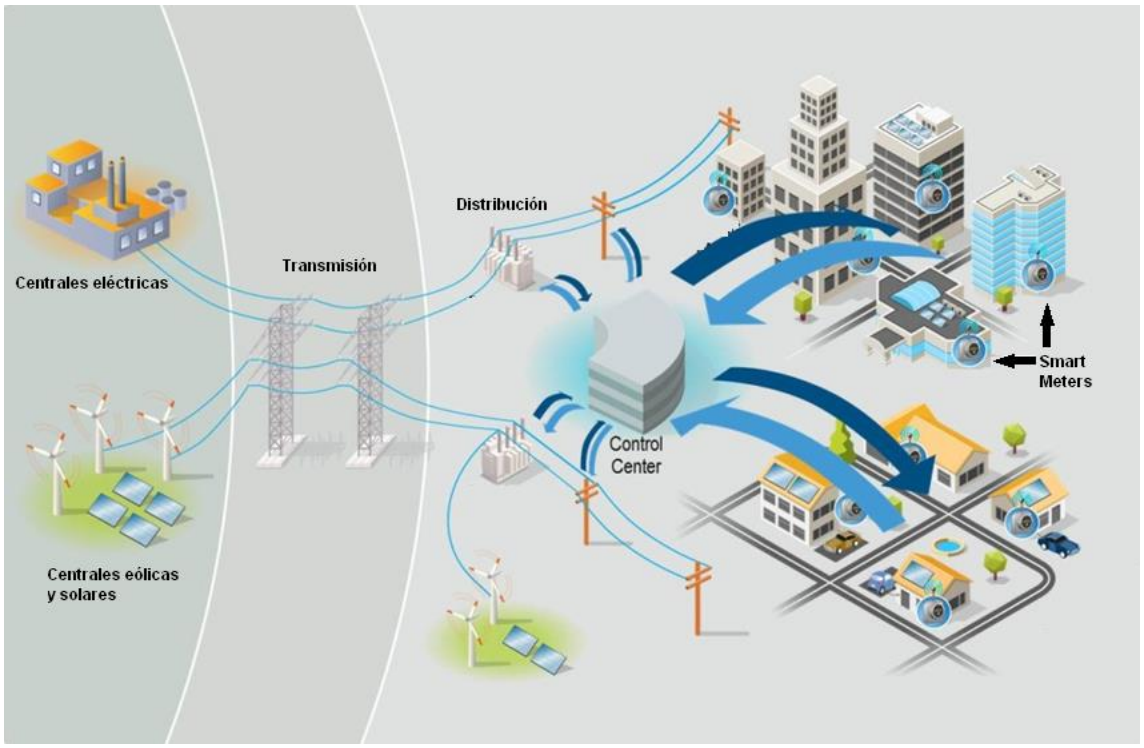

Ilustración 2.1. Esquema de una smart grid.

Con las **redes eléctricas convencionales**, no se pueden obtener prácticamente ningún tipo de datos de muestreo de las instalaciones eléctricas, ya que el flujo de esta red es solamente de energía. Por lo tanto, cualquier error en el sistema no se presenta a las compañías eléctricas hasta que un cliente se lo notifica, en la gran mayoría de las ocasiones. La información en tiempo real que se tiene con estas redes es prácticamente nula.

Una **smart grid** se puede conectar a la domótica de una vivienda e interactuar con ella para integrar de forma eficiente el comportamiento y las acciones de todos los usuarios conectados a ella, de tal forma que se asegure un sistema energético sostenible y eficiente, con altos niveles de calidad y seguridad de suministro y lo más importante, con bajas pérdidas.

Además, la **smart grid** es capaz monitorizar y medir el comportamiento eléctrico de cada uno de los aparatos que están conectados a la misma, lo que nos ofrece la ventaja de saber cuál es nuestro consumo en todo momento.

A continuación, se muestra una tabla con las principales diferencias entre una **red eléctrica convencional** y una **smart grid**:

Tabla 2.1. Comparación entre red eléctrica convencional y smart grid

	Red convencional	**Smart grid**
Comunicaciones	Ninguna o unidireccional	Bidireccional
Interacción con consumidores	Escasa o nula	Generalizada
Operación y mantenimiento	Manual	Monitorización remota
Generación de electricidad	Centralizada	Centralizada y distribuida
Medidores de energía (Contadores)	Electromecánico	Digital
Control del flujo	Limitado	Generalizado y flexible
Restablecimiento del suministro	Manual	Autorrestablecimiento
Topología	Radial	Mallada
Fiabilidad del suministro	Fallos e interrupciones	Protecciones adaptativas

2.1.1. Respuesta a la demanda

Una de las principales características de una **smart grid** es que puede gestionar el suministro de energía eléctrica de forma controlada desde las centrales de generación hasta los equipos de consumición activos. La **respuesta a la demanda** puede ofrecer una amplia gama de beneficios potenciales en la operación del sistema y en la eficiencia del mercado, debido a que promueve la interacción y la capacidad de respuesta de los clientes.

Un ejemplo claro de los beneficios de la respuesta a la demanda es eliminar la sobrecarga de la red. Con los sensores de los que dispone una **smart grid** es posible detectar los aumentos de consumo de energía eléctrica en una determinada zona. De esta manera se puede actuar antes de que se sobrecargue la red y conmutarla para reducir la demanda en esa zona específica, eliminando así el problema.

Por otra parte, la infraestructura de medición avanzada que posee una **smart grid** tiene la capacidad de analizar en tiempo real los costes de energía en una determinada franja horaria, pudiendo ampliar así el rango de programas de tarifas basadas en el tiempo que se pueden ofrecer a los consumidores.

Si, además juntamos la respuesta a la demanda con el uso de domótica, se pueden aprovechar los momentos claves en el mercado energético y, por ejemplo, programar una lavadora para que realice un lavado en el momento del día en el cual el consumo de energía eléctrica es más barato. Esto hace que sea más fácil para los consumidores cambiar su comportamiento y reducir el consumo en el período pico de demanda, reduciendo así sus costes. Además, las empresas reducen las horas donde la generación de energía eléctrica se encuentra al límite.

“El suministro eléctrico es un problema social constante por el crecimiento desproporcionado. Y los programas de respuesta a la demanda proporcionan ventajas económicas estimadas, entre el 5% y el 25%, a quien participa y ayuda a desplazar el consumo de las horas pico o críticas, donde la capacidad de generación está al límite.” [8]

2.1.2. Flujo de energía

Otra de las funcionalidades más importantes de una **smart grid** es que permite gestionar el **flujo de energía** eléctrica en ambos sentidos. Además de proporcionar un flujo de energía eléctrica al consumidor, la compañía que suministra la energía recibe un flujo de datos del cliente. La información que se recibe del consumidor permite a las compañías operar la red eléctrica de manera más eficiente.

Aprovechando esto, un consumidor puede llegar a ser proveedor de energía eléctrica si instala en su red doméstica un equipo que genere energía, como por ejemplo una célula fotovoltaica. Si el consumidor es capaz de generar su propia energía podrá devolverla a la red, recibiendo una compensación económica por ella.

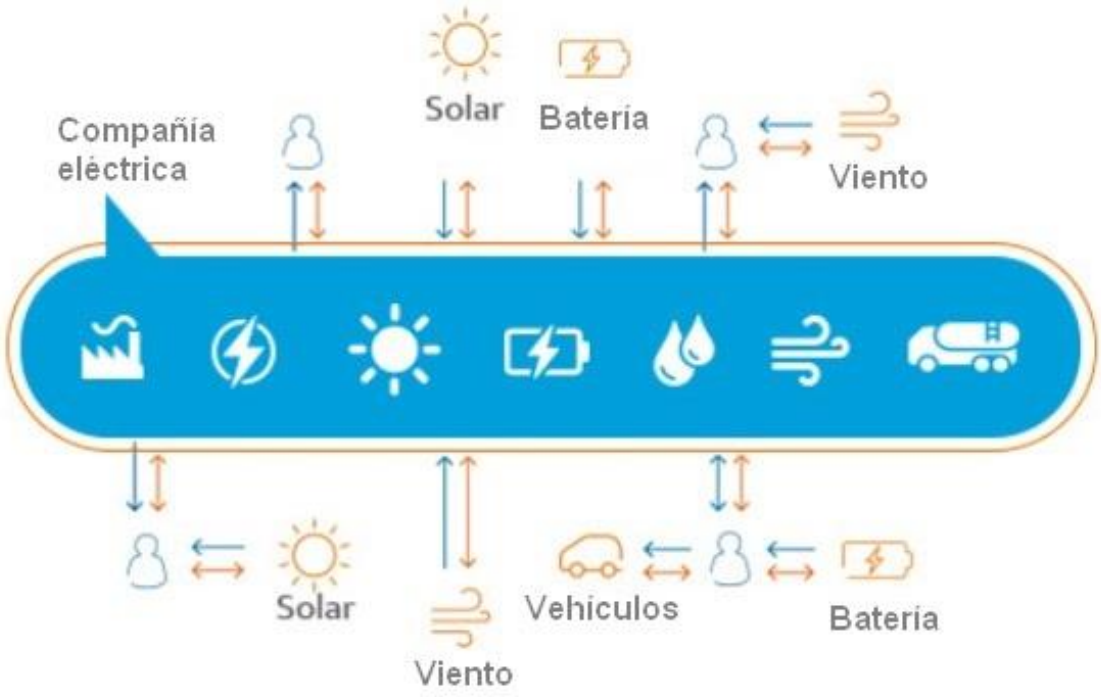

Ilustración 2.2. Flujo de energía eléctrica en una smart grid (bidireccional)

Con las **redes eléctricas convencionales**, la energía generada en las plantas eléctricas que no se consume en un breve periodo de tiempo, se pierde. En un momento como éste, en el que se busca la máxima eficiencia energética, hay que buscar soluciones. Por este motivo, las smart grids representan la transición hacia una futura versión de las redes eléctricas convencionales. Una evolución de una red centralizada a una más derivada que exija una mayor interacción por parte del consumidor.

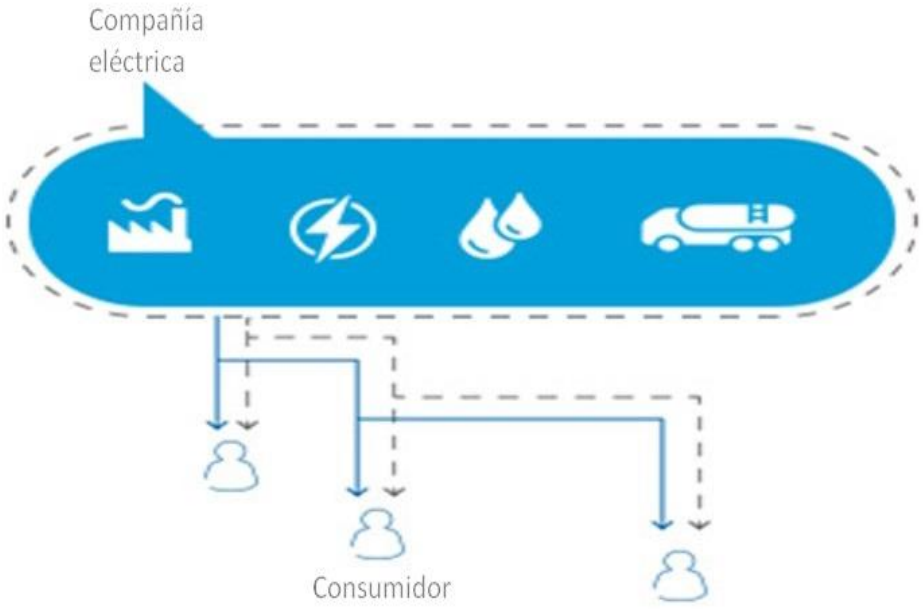

Ilustración 2.3. Flujo de energía en una red eléctrica convencional (unidireccional)

2.2. Smart meter

Un **smart meter** es un dispositivo electrónico que registra el consumo de energía eléctrica, gas o agua. Sin embargo, el término "**smart meter**" se suele asociar directamente a la **energía eléctrica**.

Cabe aclarar que los **smart meters** constituyen una pieza indispensable de la smart grid, pero ellos solos no la constituyen. Necesitamos de otros componentes (sensores, domótica, etc.) para poder implementar una smart grid.

Al formar parte de una smart grid, los **smart meters** presentan comunicación con el servidor o sistema central. Por lo tanto, un **smart meter** puede enviar toda la información registrada hasta un centro común (normalmente a la compañía suministradora de la energía), para una vez allí poder procesarla de forma conjunta con la información de otros **smart meters**.

Las comunicaciones con el **smart meter** pueden ser inalámbricas (Wi-Fi y Wi-SUN) o a través PLC (del inglés Power Line Carrier), que utiliza el cableado de la propia red eléctrica para establecer la comunicación. Esta última es la utilizada para implementación de smart grids.

En la ilustración 2.4 se muestra el modelo de smart meter con el que se ha trabajado en este proyecto.

Ilustración 2.4. Smart meter [27]

A diferencia de un contador eléctrico convencional, el **smart meter** calcula el consumo de una forma más detallada, especificando el consumo individualizado de todos los aparatos conectados a él y permitiendo así un mayor control de la gestión de energía por parte del usuario.

Además, tienen la capacidad de poder interrumpir el suministro de energía de forma remota, y configurar a medida del usuario el tipo de servicio o tarifa de facturación de la energía.

.

2.3. Protocolo DLMS/COSEM

DLMS/COSEM [24] es un conjunto de estándares que especifica un **protocolo de comunicación** bidireccional y un **modelo de interfaz** para la conexión con **smart meters** de cualquier tipo de energía: electricidad, gas, agua o calor. Fue desarrollado en 1997 y proporciona un medio para para la comunicación con **smart meters** a través de un intercambio de datos estandarizados.

DLMS (Device Language Message Specification) es una especificación de capa de aplicación diseñada para admitir y establecer la comunicación y poder acceder a los objetos.

COSEM (COmpanion Specification for Energy Metering) es el **modelo de interfaz** que se utiliza en la comunicación para intercambiar datos. Es un modelo que utiliza un enfoque orientado a **objetos**.

Es el protocolo de comunicación más popular utilizado en el ámbito de la medición en todo el mundo. Sus principales aplicaciones son la lectura remota de **smart meters**, su control remoto y proporcionar los servicios para medir cualquier tipo de energía. Ha sido aceptado por la mayoría de los servicios públicos de energía a raíz del mercado de energía liberalizado.

Los protocolos de este estándar se definen en un conjunto de **cuatro documentos** de especificaciones:

- El **libro azul** describe el modelo de interfaz de modelado de objetos del protocolo DLMS/COSEM y el sistema de identificación de objetos OBIS. [24]

- El **libro verde** describe la arquitectura y los protocolos.

- El **libro amarillo** recopila todas las pruebas de conformidad del protocolo DLMS/COSEM para probar y certificar las especificaciones descritas en el libro azul. [18]

- El **libro blanco** contiene el glosario de términos.

Si un producto pasa la prueba de conformidad especificada en el **libro amarillo**, la asociación de usuarios de DLMS emite una Certificación de conformidad de dicho producto con DLMS / COSEM.

Por último, cabe señalar que este protocolo además de poder recopilar datos en diferentes formatos, también se adapta a las necesidades futuras, como la adición de más funcionalidades. Esto resalta la importancia de este protocolo debido a que en el mercado de la energía eléctrica se requiere de una gran cantidad de información de los medidores de energía para que las estimaciones de gasto y el proceso de facturación sean lo más certero posible.

2.3.1. Modelo de interfaz

El protocolo DLMS/COSEM [24] sigue un modelo de datos basado en técnicas de **modelado de objetos**, incluyendo interfaces y **clases de objetos**. Su modelo de interfaz presenta una estructura jerárquica de tres niveles:

1. Dispositivo físico
2. Dispositivo lógico
3. Objetos

1) Dispositivo físico

El dispositivo físico es el smart meter. El smart meter tiene una única dirección a nivel hardware. Esta dirección se conoce como MAC (del inglés Media Access Control), la cual tiene seis bytes de longitud.

Dentro de él, un smart meter puede alojar uno o varios dispositivos lógicos. Además, el smart meter contiene al menos un dispositivo lógico de gestión que tiene la descripción de todos los dispositivos lógicos que tiene disponibles, con sus direcciones lógicas y nombres.

2) Dispositivo lógico

El **dispositivo lógico** especifica el tipo de medidor que posee dentro un smart meter. Por ejemplo, en un smart meter de energía múltiple, en un dispositivo lógico podría ser un medidor de electricidad, otro, un medidor de gas, etc.

El **dispositivo lógico** contiene en su interior **todos los objetos**, con sus correspondientes atributos y métodos, que puede albergar el medidor.

En este TFG se ha trabajado con un smart meter que posee únicamente un dispositivo lógico, que mide energía eléctrica.

3) Objetos

Las clases de interfaz se basan en la estructura de programación **orientada a objetos**:

- Los objetos son la instancia de una clase y están provistos de un conjunto de **atributos y métodos**.

- Los **atributos** son las características que tiene la clase, y en función de su valor, puede modificar el comportamiento del objeto.

- Los **métodos** son las funciones asociadas a un objeto, que consecuentemente reaccionan a eventos que se producen de manera externa. Además, los métodos permiten realizar operaciones en atributos.

Los objetos que poseen características similares se agrupan en las mismas clases. El primer atributo de cualquier objeto es el **nombre lógico**, el cual lo identifica de forma inequívoca. El valor del nombre lógico consiste en un conjunto numérico de seis valores definidos según el sistema de identificación **OBIS**.

2.3.2. Códigos OBIS

OBIS (del inglés Object Identification System) es un sistema de identificación que permite determinar el valor de los nombres lógicos de los objetos contenidos dentro de un smart meter. Cada código **OBIS** identifica un único objeto de forma inequívoca. Los códigos **OBIS** se usan para la identificación de:

- Nombres lógicos
- Datos mostrados en la pantalla del smart meter
- Datos transmitidos a través de las líneas de comunicación

La estructura general de los códigos **OBIS** está compuesta por **seis valores** (A, B, C, D, E, F). Cada valor corresponde a un byte, con un **número del 0 al 255**. Cada valor tiene su propio rango y una definición que hace posible identificar los distintos objetos que contiene el smar meter.

En la tabla 2.2 se muestra la descripción de cada valor. Y en *la tabla 3.3* se muestra la correspondencia de valores del valor A.

Tabla 2.2. Descripción de los distintos valores de un código OBIS [9]

Grupo	Descripción
A	Especifica el medio de energía en cual se mide el objeto (ver tabla 2.3)
B	Identifica el número de canal de entrada. Un smart meter puede poseer múltiples canales de entrada para los diferentes tipos de datos
C	Define la magnitud física o abstracta de medida del objeto (voltaje, potencia, temperatura, volumen, etc). La definición depende del valor del grupo A
D	Define los tipos o el resultado del procesamiento de las cantidades físicas identificadas con el valor de los grupos A y C de acuerdo con una serie de algoritmos.
E	Define la clasificación adicional de los resultados de medición identificados con los grupos A, B, C y D para los smart meters, dependiendo de la tarifa contratada
F	Define el almacenamiento de datos, identificado por los grupos de valores de A hasta E, de acuerdo con diferentes periodos de facturación. Cuando esto no sea relevante, este número toma valor 255.

Tabla 2.3. Definición de valores del byte A de los códigos OBIS [9]

Código	Definición
0	Objetos abstractos
1	Objetos de electricidad
4	Objetos de distribución de costes
5	Objetos de medida de frío
6	Objetos de medida de calor
7	Objetos de gas
8	Objetos de agua fría
9	Objetos de agua caliente
Cualquier otro	Reservado para uso futuro

La asociación de usuarios de DMLS [28] mantiene regularmente una lista de códigos OBIS estándar y objetos COSEM.

Por convenio, los distintos fabricantes de smart meters están obligados a utilizar siempre los mismos nombres lógicos para referirse a los mismos objetos. De esta manera, cualquier objeto es accesible a través de su nombre lógico con independencia del modelo o fabricante que haya creado el smart meter, permitiendo así estandarizar las lecturas de objetos de cualquier smart meter.

Cabe destacar que los diferentes fabricantes de smart meters pueden implementar códigos OBIS que no sean estándar, pero de hacerlo, no pueden reusar códigos OBIS estandarizados para identificar otros objetos con distinto significado. Los códigos que desarrollan los fabricantes pueden ser posteriormente estandarizados si son de interés común para todos los demás modelos de smart meter.

2.4. IEC 62056

Para la medición de energía eléctrica, el protocolo DLMS/COSEM tiene una versión específica basada en la serie **IEC 62056**. **IEC 62056** es un conjunto de estándares para **medición de electricidad**, intercambio de datos para lectura de smart meters, control de tarifas y control de carga, establecido por la Comisión Electrotécnica Internacional (IEC) [10].

IEC 62056 está estructurado en función de un modelo cliente-servidor en el que el cliente realiza las solicitudes y el servidor (el smart meter) devuelve los datos correspondientes.

Las normas **IEC 62056** están enfocadas en la medición de electricidad, mientras que DLMS/COSEM abarca un campo más general, y se aplica a cualquier medición de energía.

IEC 62056-21

Esta parte del estándar IEC 62056 describe las especificaciones de hardware y protocolo para el intercambio de datos con un contador eléctrico de forma local, mediante conexión física con una unidad de mano. La conexión se realiza mediante un **acoplamiento óptico** con el smart meter.

El protocolo permite la lectura y programación del smart meter y está diseñado de manera adecuada para el entorno de medición de electricidad, con especial atención al aislamiento eléctrico y la seguridad de los datos. Si bien el protocolo está totalmente definido, el uso y la aplicación de este, sobre un smart meter, debe realizarlo el usuario.

En la relación cliente-servidor, la unidad de mano (cliente) que establece conexión con el smart meter actúa como maestro, mientras que el propio smart meter actúa como esclavo.

2.4.1. Sonda óptica

Para poder establecer la conexión de forma física entre el smartphone y el smart meter es necesario el uso de una **sonda óptica modelo KMK 116** [11].

La sonda óptica tiene una terminación en USB y con un adaptador de USB a micro USB es posible conectarla al smartphone. La conexión se establece siguiendo el **estándar** de conexión **IEC 62056-21**.

Ilustración 2.5. Sonda óptica utilizada en este TFG

Propiedades físicas:

- Tipo de señal: bucle de corriente de 20 mA
- Valores límites:
    - Tensión de circuito abierto máxima: 30 V CC
    - Corriente de bucle máxima: 30mA

Tabla 2.4. Características eléctricas de la sonda óptica [12]

Corriente	**Emisor (TX)**	**Receptor (RX)**
Sin corriente de bucle	≤ 2,5 mA	≤ 3 mA
20 mA de corriente de bucle	≥ 11 mA	≥ 9 mA
Tensión	**Emisor (TX)**	**Receptor (RX)**
20 mA de corriente de bucle	≤ 2 V	≤ 3 V

El smartphone suministra la energía necesaria a la sonda óptica cuando esta se encuentra conectada a él.

En la *ilustración 2.6*, se muestra el esquema de conexión entre el smartphone y el smart meter, a través de la sonda.

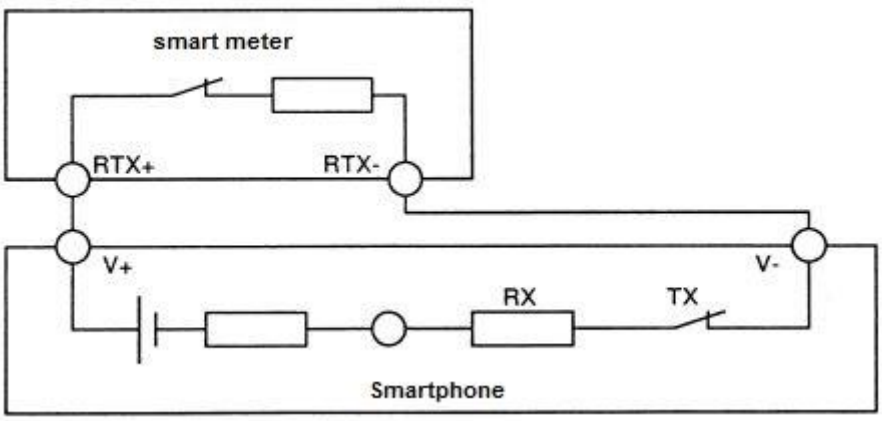

Ilustración 2.6. Esquema de conexión del smartphone con el smart meter [12]

La transferencia de datos óptima se logra cuando el cabezal de la sonda lectura está en la posición correcta, con el cable hacia abajo, sobre el receptor de infrarrojos del que dispone el smart meter.

Además, la sonda óptica dispone de un imán en su cabezal, *véase la ilustración 2.7*, que genera un campo magnético sobre el cabezal óptico del smart meter, facilitando su colocación y encaje sobre él, y dificultando así su movimiento una vez esta se sitúa.

Las variaciones leves en esta posición no deberían afectar significativamente el rendimiento, pero para variaciones más grandes, puede llegar a producir una degradación de la calidad de conexión.

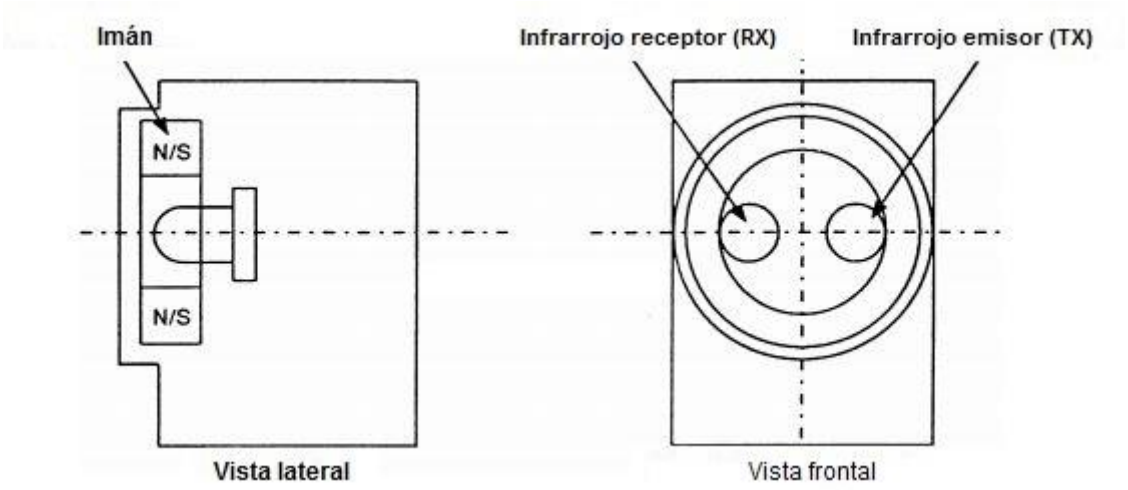

Ilustración 2.7. Dibujo esquemático de la sonda [12]

2.5. Android Studio

Android Studio [13] es un entorno integrado de desarrollo integrado (en inglés, IDE, Integrated Development Environment), basado en IntlliJ, para la plataforma Android. En otras palabras, es una herramienta que facilita la labor del programador ofreciéndole ayuda para desarrollar aplicaciones para teléfonos Android.

Android Studio ha sido desarrollado por Google y es actualmente el único programa oficial de Android que da soporte. El lenguaje de programación en el que está basado es **Java**.

2.5.1. Activities en Android

Una **activity** es un componente de las aplicaciones de Android que provee una **interfaz gráfica** a través del dispositivo, permitiendo así que el usuario pueda interactuar con la aplicación. En un lenguaje más común las **activities** son cada una de las pantallas que se muestran en la ejecución de una aplicación.

Cada **activity** se compone de una parte gráfica y de una parte lógica. La **parte gráfica** consiste en un layout que se programa en XML. En esta parte se diseña la interfaz que se va a mostrar al usuario.

Por otro lado, la **parte lógica** consiste en una clase de **Java** que permite programar los ciclos de vida de nuestra **activity**, así como todas las acciones que debe realizar la aplicación en función de cómo el usuario final interactúe con la pantalla del dispositivo.

Las **activities** pasan por diferentes estados a lo largo de la ejecución de una aplicación en un dispositivo:

- **Activo**: En este estado la **activity** se encuentre la primera en la pila de ejecución. En este estado el usuario la ve por pantalla y puede interactuar con ella.

- **Pausa**: En este estado la **activity** pasa a segundo plano, pero aún está disponible en la memoria del dispositivo, no se ha eliminado. Esto ocurre

cuando otra actividad se sitúa en pantalla, y la anterior desparece de la vista del usuario. En este caso, la actividad “tapada” puede ser cerrada por el sistema si necesita liberar recursos para la nueva actividad.

- **Parada**: Es similar al estado Pausa. La principal diferencia es que si la **activity** vuelve al primer plano se vuelven a ejecutar las tres funciones para refrescarla. Además, si el dispositivo necesita liberar memoria, liberará una **activity** que este en estado de **parada** antes que una **activity** que se encuentre en estado de **pausa**.

En la ilustración 2.8 se puede observar el ciclo de vida de una **activity** y como se pasa de un estado a otro.

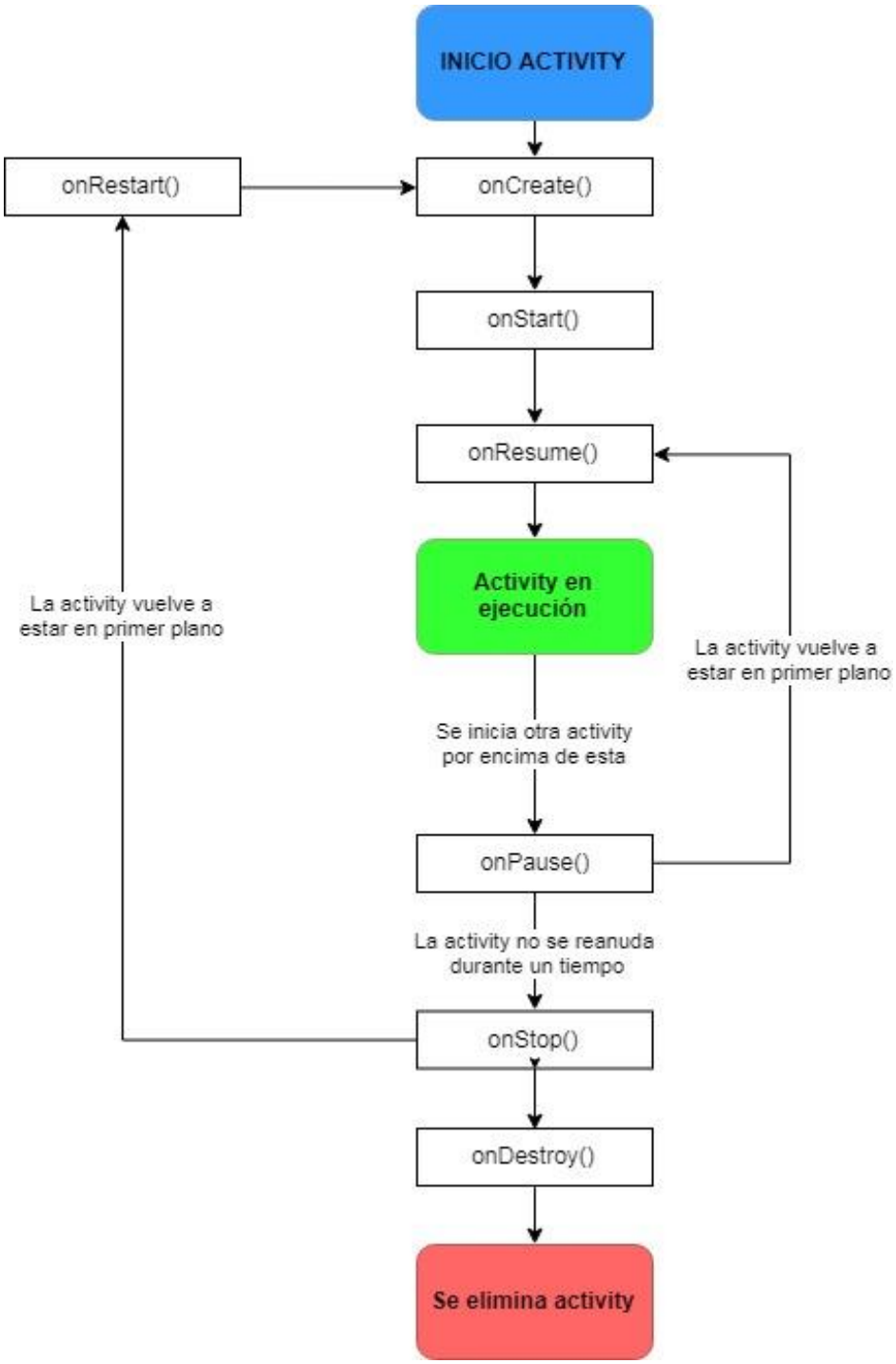

Ilustración 2.8. Ciclo de vida de una Activity

A continuación, se detallan de forma cronológica los métodos de gestión del ciclo de vida de una activity:

- **onCreate**: Este método se llama al crear la actividad. Es donde se prepara la interfaz gráfica de la pantalla. Tras esta función, el proceso sobre el que se ejecuta la Actividad no puede ser destruido por el sistema.

- **onStart:** Este método se ejecuta después del método **onCreate** y justo antes de que la aplicación sea visible al usuario.

- **onResume:** Este método se ejecuta en el momento en el cual la actividad se encuentra visible por el usuario, pero justo antes de que pueda interactuar con ella.
  Al ejecutarse este método, la activity pasa a estado **Activo**.

- **onPause:** Este método ejecuta justo antes de que la actividad pase a ser tapada por otra nueva. En este se debe aprovechar para liberar todo aquello que consume o para guardar datos de manera persistente. No puede contener tareas lentas ya que hasta que no termine este método no podrá ejecutarse el **onResume** de la nueva actividad.
  Al ejecutarse este método la activity pasa a estado **Pausa**.

- **onStop:** Este método se ejecuta cuando la actividad pasa a segundo plano, es decir, cuando se hace invisible al usuario. Si después la actividad vuelve a primer plano, el siguiente método que se ejecutará será onRestart.
  Al ejecutarse este método la activity pasa a estado **Parada**.

- **onDestroy**: Este método se llama justo antes de destruir la actividad. El motivo de la llamada puede ser porque se ejecute el método de finalización de actividad (**finish**) o porque el sistema elimine de forma autónoma la actividad para conseguir más recursos. Durante la destrucción de la actividad se perderán todos los datos asociados a ella por lo que en este método podrá ser utilizado para controlar la persistencia de datos.

2.6. HTTP

HTTP (del inglés HiperText Transfer Procotol) es un **protocolo** estándar internacional que permite la **transferencia de datos** e información a través de internet. Está basado en una serie de reglas transferencia de recursos o archivos entre equipos que se encuentran conectados a una misma red.

Es un protocolo orientado a transacciones y sigue el esquema petición-respuesta entre dos equipos. El equipo que hace la petición para obtener datos o enviarlos, se le denomina **cliente** y el que tiene el recurso o contiene el espacio para almacenar se le denomina **servidor**.

Ilustración 2.9. Esquema conexión HTTP [14]

La comunicación se establece a través de una petición de envío, la cual contiene todos los datos del cliente, como el navegador web desde donde se realiza la petición, el sistema operativo que utiliza o la ubicación del archivo solicitado (URL).

HTTP tiene ya definidos una serie de métodos de petición, los cuales se han ido añadiendo según se desarrollaban nuevas versiones, debido a que el protocolo tiene flexibilidad para añadir nuevos métodos que aumenten el número de funcionalidades que ofrece. Los métodos más frecuentemente utilizados son:

- Método **GET** (Retorno de datos): En este método el cliente solicita al servidor un recurso. Esta solicitud solo devuelve datos. Un ejemplo claro de este método se produce al realizar una búsqueda en la web, donde el cliente especifica la URL y el servidor devuelve la información necesaria para que el represente por pantalla la página web solicitada.

- Método **POST** (publicación de datos): En este método el cliente envía los datos necesarios al servidor para que este lo almacene en su base de datos. Este método es el opuesto al método GET. Un ejemplo claro de este método se produce al rellenar un formulario para una página web.

En la aplicación que se ha desarrollado en este proyecto, se utiliza un método **POST** para enviar los datos leídos del smart meter a un servidor.

3. DESARROLLO DE LA APLICACIÓN

En este epígrafe del trabajo se desarrolla el procedimiento de **conexión entre el smartphone y el smart meter** que se ha llevado a cabo en la programación de la aplicación de Android, además del **proceso de lectura de los objetos del smart meter**, sin mostrar interfaces gráficas (Activities) de la misma.

Más adelante, en el **epígrafe 5**, se muestran las interfaces de la aplicación, enseñando cómo funcionan las Activities de esta y como se hace uso de los métodos que se van a exponer a continuación.

Se ha tomado la decisión de dividir así la estructura de la memoria debido a que la dificultad de aplicación reside en la programación interna de las clases que establecen conexión con el smart meter, mientras que interfaces gráficas son sencillas, al tratarse de una aplicación que automatiza la lectura del smart meter.

3.1. GuruX

GuruX Ltd. [15] es una organización que ofrece una colección de componentes de software en código abierto que permiten el establecimiento de conexión con smart meters, principalmente para la comunicación DLMS con medidores de electricidad, agua o gas. El código de **GuruX** contiene implementaciones en C#, C++, Java, Delphi y Python.

Para desarrollar esta aplicación en Android, se ha importado una **librería de clases de Java** que proporciona el conjunto de clases necesarias para establecer la comunicación con un smart meter a través del **protocolo DLMS/COSEM**, además de las clases de los objetos del modelo de interfaz. La versión que se ha utilizado para esta aplicación es la **número 22**. [16]

También se ha importado otra librería [17] de **GuruX** que ofrece las clases necesarias para enviar y recibir datos a través del puerto microUSB de los dispositivos **Android**. Esto permite crear un sistema de lectura automática de contadores, **AMR** (del inglés **Automatic Meter Reading**), para gestión de consumo eléctrico, a través de conexión en serie mediante sonda óptica.

3.2. Conexión cliente-servidor

A lo largo de este epígrafe, se hará referencia al smartphone como **cliente** y al smart meter como **servidor**.

Todo el proceso de establecimiento de conexión entre el cliente y el servidor se realiza con la clase **GXDLMSClient** de la librería de **GuruX** para Java.

Para establecer una comunicación entre el par **cliente-servidor** es necesario crear una **Aplicación de Asociación** (**AA**) entre ambos. Para ello, el **cliente** es el que se debe de encargar de iniciar la conexión para establecer la comunicación.

El punto de partida para establecer una comunicación DLMS/COSEM es la solicitud **SNRM** (del inglés Set Normal Response Model). Esta solicitud transmite la intención del cliente de establecer una conexión con el servidor. El cliente la envía y recibe una respuesta **UA** (del inglés Unnumbered Acknowledge) por parte del servidor. Entre estas dos operaciones se negocian los parámetros de conexión para lograr un intercambio de datos de forma exitosa.

El siguiente par de mensajes consiste en la solicitud de Asociación de Aplicación, **AARQ** (del inglés Application Association Request) y la respuesta de Asociación de Aplicación, **AARE** (también del inglés Application Association Response).

AARQ es un comando que envía el cliente al servidor y en el que indica que tipo de autenticación se usa para la conexión y que tipo de referencia se utiliza. En el caso concreto de la aplicación desarrollada, la autenticación escogida ha sido de nivel bajo (usuario y contraseña) y el tipo de **referencia** a objetos se realiza a través de nombres lógicos, mediante **códigos OBIS.**

Una vez que el servidor recibe el **AARQ**, analiza la petición, configura el modelo de interfaz y establece una **Asociación de Aplicación** caracterizando el contexto de la comunicación entre el cliente y el servidor. Finalmente, el servidor devuelve, a través del comando **AARE**, una colección de etiquetas.

En la ilustración 3.1 se representa un esquema simplificado del flujo de intercambio de mensajes que se acaba de explicar.

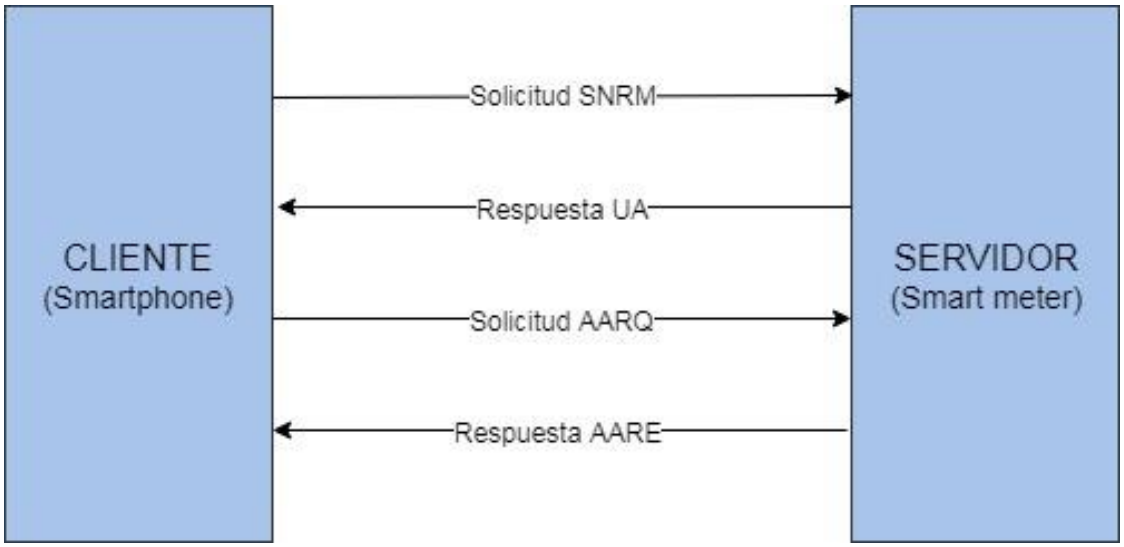

Ilustración 3.1. Esquema de establecimiento conexión cliente-servidor

Después del establecimiento de la conexión, todas las solicitudes de intercambio de información entre el smart meter y el smartphone utilizan **códigos OBIS**, códigos estandarizados para todos los dispositivos que cumplen con el protocolo DLMS/COSEM.

3.3. Association View

Una vez que se ha establecido la conexión entre el **servidor** (smart meter) y el **cliente** (smartphone) resulta necesario conocer todos los objetos que contiene el meter antes de realizar ninguna lectura.

La cantidad de objetos de un meter depende del modelo y del fabricante, ya que como se ha comentado en anteriores epígrafes de este trabajo, una de las funcionalidades principales del modelo de interfaz del protocolo DLMS/COSEM es que permite actualizarse integrando nuevos objetos según sean necesarios,

La integración de un nuevo objeto se producirá siempre y cuando supere las pruebas de conformidad especificadas en el libro amarillo. [18]

Una vez que se establece la conexión entre el smartphone y el smart meter, la primera solicitud que se realiza es la lectura del **Association View**. El Association View es un objeto que proporciona al cliente el acceso al **modelo de interfaz** de objetos del smart meter. Esta lectura devuelve una lista de todos los objetos disponibles dentro del dispositivo lógico del smart meter.

Además, el **Association View** contiene, entre otros atributos, la descripción de cada objeto, que es fundamental para saber la funcionalidad de cada objeto.

La clase **GXDLMSObjectCollection** de la librería de **GuruX** permite realiza una solicitud para leer el **Association view**. Como todo proceso de lectura de objetos, la lectura se realiza enviando el código OBIS del objeto a leer. En este caso el código OBIS del **Association View** es el **0.0.40.0.0.255**

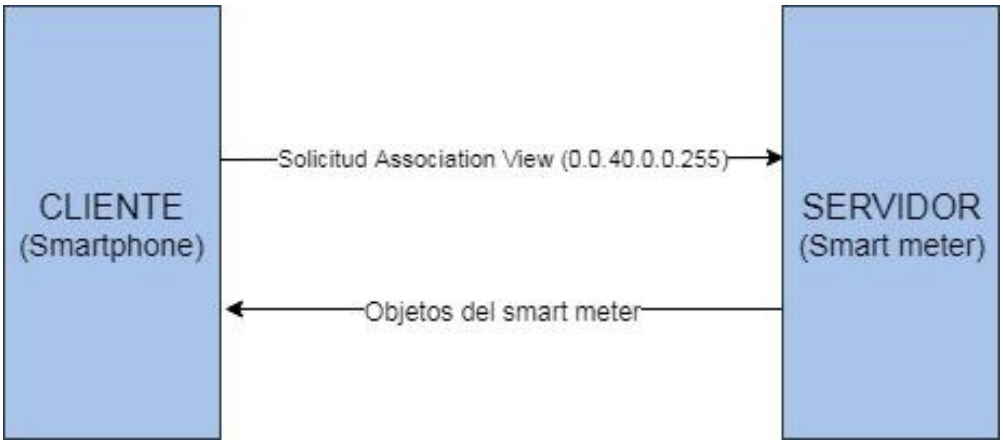

Ilustración 3.2. Intercambio mensajes en la lectura del Association View

La lectura con la clase **GXDLMSObjectCollection** devuelve un array que contiene **todos los objetos** disponibles en el smart meter.

Para saber la funcionalidad de cada objeto necesitamos conocer la descripción de cada objeto. La clase **GXDLMSConverter** nos permite obtener la descripción de todos los objetos del **Association View**.

3.4. Lectura de objetos a través de código OBIS

Una vez que sabemos los objetos de los que dispone el meter es posible tener acceso a cada objeto, ya sea para leerlos o para modificar sus parámetros.

Todos los objetos **se leen según el mismo procedimiento**, siempre a través de su **código OBIS**. En la ilustración 3.3 se muestra a modo ejemplo de la lectura del objeto **GXDLMSClock** que es el objeto que establece la fecha y hora interna del smart meter.

```
public String leerReloj() throws Exception {

    GXDLMSClock object = new GXDLMSClock();
     object.setObjectType(ObjectType.CLOCK);
     object.setLogicalName("0.0.1.0.0.255");
     object.setUIDataType(2, DataType.DATETIME);

     Object o = com.readObject(object, 2);

     String lectura = com.readValue(o);
     return lectura;
}
```

Ilustración 3.3. Lectura del objeto Clock del smart meter

Esta forma de lectura sirve igual para todos los objetos, cambiando claro está los parámetros en función del objeto a leer:

- Se instancia la clase de **GuruX** correspondiente al objeto a leer.
- Se configura el atributo **ObjectType**.
- Se configura su nombre lógico según el código **OBIS**.
- Se configura el atributo **DataType**.
- Se llama a la función **readObject**
- Por último, se llama a la función **readValue** que devuelve un String con la información específica del objeto leído.

4. RESULTADOS OBTENIDOS

4.1. XML del Association View

El array con todos los objetos que se ha obtenido con la clase **GXDLMSObjectCollection** (*ver epígrafe 3.3*) consiste en un conjunto de información en formato binario. Para imprimirlos y saber cuáles son los objetos con los que se pueden trabajar en este proyecto, es necesario traducir los datos a un lenguaje legible.

Se ha optado por la opción de convertir toda la información de bytes a texto con una **estructura de XML**. Para ello se ha creado un fichero en el que, de forma ordenada, se escriben uno a uno **todos los objetos** que contiene smart meter, cada con sus principales atributos en función del tipo de objeto.

Para crear el fichero XML se ha hecho uso de la clase **XmlSerializer** [19]. Esta clase, proporcionada por la librería de Android por defecto, permite crear de forma sencilla un String de texto con **estructura XML**.

En la ilustración 4.1 se muestra un extracto del fichero **XML** del Association View que se ha obtenido una vez traducido el array de objetos. Aunque el fichero obtenido cuenta con 486 objetos, sin embargo, solo se muestran tres objetos a modo de ejemplo.

```xml
<?xml version='1.0' encoding='UTF-8'?>
<Objects>
    <GXDLMSClock>
        <LN>0.0.1.0.0.255</LN>
        <Description>Ch. 0 Clock object  #1</Description>
    </GXDLMSClock>

    <GXDLMSRegister>
        <LN>1.1.94.34.1.255</LN>
        <Description>Ch. 1 Identifiers for Spain</Description>
        <Unit>7</Unit>
        <Scaler>1.0</Scaler>
    </GXDLMSRegister>

    <GXDLMSData>
        <LN>1.1.94.34.104.255</LN>
        <Description>Ch. 1 Identifiers for Spain</Description>
    </GXDLMSData>
</Objects>
```

Ilustración 4.1. XML del Association View

Cada objeto tiene diferentes atributos, comunes o específicos del objeto en concreto. Para crear el fichero, se han seleccionado una serie de atributos claves y comunes para identificar cada objeto. Estos atributos son: **Descripción** y **nombre lógico**.

- **Descripción:** Proporciona una breve explicación para identificar el objeto.
- **Nombre lógico:** Proporciona el código OBIS del objeto, necesario para acceder a él.

Además, existen objetos especiales, de los cuales se han leído más atributos, a parte de los comunes, ya que son específicos del objeto en concreto y se ha creído importante su lectura. Estos objetos son: **Register**, **Extended Register** y **Profile Generic**.

- **Register**: Este objeto se utiliza para guardar un valor con una escala y unidad asociados. Aparte de los atributos comunes, en este objeto se ha leído:

    - **Unidad**: Indica la unidad de medida mediante el un número, el cual equivale a una unidad del Sistema Internacional. *En la tabla 4.1.* se han indicado las principales unidades de medidas en función de su valor.

    - **Escala**: Indica la escala de la unidad de medida.

Tabla 4.1. Unidades de medida en función del número [16]

Número	Unidad	Nombre
1	tiempo	año
2	tiempo	mes
4	tiempo	día
5	tiempo	hora
6	tiempo	minuto
7	tiempo	segundo
25	J	Energía
27	W	Potencia activa (P)
28	VA	Potencia aparente (S)
29	vaR	Potencia reactiva (Q)
33	A	Corriente (I)
34	C	Carga eléctrica (Q)
35	V	Voltaje (V)
37	F	Capacitancia(C)
38	Ω	Resistencia
39	Ω•m	Resistividad (ρ)

- **Extended Register**: Este objeto también se utiliza para guardar un valor con su escala y unidad. Sin embargo, se diferencia del objeto **Register** en que, además, indica la frecuencia con la que se registran valores.

    - **Unidad**: Indica la unidad de medida mediante el un número, el cual equivale a una unidad del Sistema Internacional. *En la tabla 4.1.* se han indicado las principales unidades de medidas en función del número.

    - **Escala**: Indica la escala de la unidad de medida.

    - **Período de captura**: Indica, en segundos, la frecuencia con la que se registra valores.

```
<GXDLMSExtendedRegister>
    <LN>1.0.1.6.10.255</LN>
    <Description>Ch. 0 Sum Li Active power+ (QI+QIV) Max. 1 Rate 10 (0 is total)</Description>
    <Unit>27</Unit>
    <Scaler>1.0</Scaler>
    <CapturePeriod>3600</CapturePeriod>
</GXDLMSExtendedRegister>
```

Ilustración 4.2. Estructura de Demand Register en XML

4.2. Profile Generic

Un **Profile Generic** es un objeto del modelo de interfaz que posee atributo con estructura en forma de matriz, que engloba los diferentes objetos que el **Profile Generic** registra, los cuales comparten una misma estructura.

El atributo en forma de matriz se denomina **CaptureObjects**, donde cada fila corresponde a un objeto capturado y cada columna corresponde al conjunto de atributos del objeto capturado.

En el **Association view**, en los **Profile Generic** se ha optado por recopilar como atributo, a parte de su nombre lógico y la descripción, la matriz **CaptureObjects**.

En la ilustración 4.3. se muestra un ejemplo de un **Profile Generic** en la que se puede apreciar como dentro del atributo **CaptureObjects** aparecen nuevos objetos (ítems), cada uno con su **nombre lógico**, **descripción**, y **ObjectType**.

```
<GXDLMSProfileGeneric>
    <LN>0.0.99.98.0.255</LN>
    <Description>Ch. 0 Event log  #1</Description>
    <CaptureObjects>
        <Item>
            <ObjectType>8</ObjectType>
            <LN>0.0.1.0.0.255</LN>
            <Attribute>9</Attribute>
            <Description>Ch. 0 Clock object  #1</Description>
        </Item>

        <Item>
            <ObjectType>1</ObjectType>
            <LN>0.0.96.11.0.255</LN>
            <Attribute>2</Attribute>
            <Description>Ch. 0 Event code  #1</Description>
        </Item>
    </CaptureObjects>

    <CapturePeriod>0</CapturePeriod>
    <SortMethod>2</SortMethod>
</GXDLMSProfileGeneric>
```

Ilustración 4.3. Estructura de un Profile Generic en XML

Para capturar objetos, un **Profile Generic** dispone de un **buffer** que los almacena. El **buffer** tiene una memoria limitada que permite almacenar un número finito de objetos capturados. No existe una capacidad predefinida para el **buffer**, la cantidad máxima de objetos que puede capturar depende del buffer de cada **Profile Generic**.

El funcionamiento del **buffer** es estático, es decir, cuando el buffer alcanza su capacidad máxima y necesita registrar un nuevo objeto capturado, elimina del registro el objeto más antiguo. Es decir, los almacena siguiendo el método **FIFO** (del inglés First In First Out).

4.3. Eventos

Un smart meter tiene la funcionalidad de poder **registrar eventos** que se generan al actuar con o sobre él mientras está en funcionamiento, conectado a la red. Además, como dispone de un reloj interno, es capaz de registrar cada evento en la **fecha y hora** en la que se produce.

Los eventos registrados dentro de un smart meter se dividen en **cinco grandes grupos**. En la tabla 4.2. se muestran los cinco grupos y los tipos de evento que se registran en cada uno.

Tabla 4.2. Tipos de evento según su clasificación [24]

Grupo	Descripción
1	Estándar
2	Control de desconexión
3	Calidad
4	Fraude
5	Operaciones de seguridad

Todos los eventos que genera el smart meter se registran dentro de un **Profile Generic**, siendo cada evento un objeto capturado en su atributo **CaptureObjects**. Cada evento que se registra tiene un **código único** que identifica de forma inequívoca la acción que lo ha generado.

Como se ha explicado en el epígrafe 4.1, cada **Profile Generic** tiene un buffer con capacidad de memoria diferente, independiente de cada uno, para almacenar objetos. Por lo tanto, cada **Profile Generic** puede **registrar un número máximo de eventos** diferente.

El smart meter con el que se han hecho pruebas tiene **once Profiles Generic** de registro de eventos. El nombre lógico cada uno sigue el siguiente formato: **0.0.99.98.X.255**, siendo **X** un número que **varía del 0 al 11** en función del tipo de eventos que registre.

A continuación, se enumeran de forma detallada los **Profile Generic de eventos con los que se han hecho pruebas** en este proyecto, y con los cuales se han comprobado que los eventos registrados se generan por la acción indicada en y la fecha y hora correcta.

Cabe aclarar que solo se ha podido probar una pequeña parte del global de todos los eventos, debido a que se ha trabajado con un smart meter que no estaba instalado en una red eléctrica, es decir, estaba aislado. Por lo tanto, no registraba toda la información de consumos y gestión de electricidad que sí registraría funcionando como contador eléctrico en una casa.

4.3.1. Eventos Estándar

El **nombre lógico** del Profile Generic que registra los eventos estándar es el **0.99.98.0.255**. Contiene todos los eventos relacionados con: actualización de firmware, finalización de los períodos de facturación, cambios de reloj, cambios de configuración, borrado de perfiles, todo tipo de errores de autocomprobación, alarmas, etc.

Estructura: Fecha y hora | Código de evento

Tamaño buffer: 100 registros.

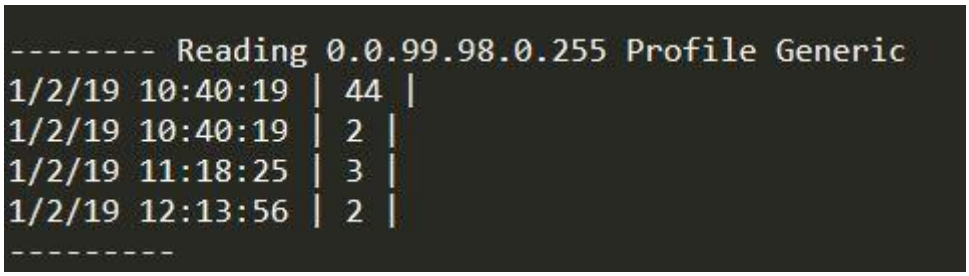

Ilustración 4.4. Eventos del Profile Generic 0.0.99.98.0.255

En la ilustración 4.4. se pueden observar los eventos registrados. El código de **evento número 2** indica que el smart meter se puso en funcionamiento, alimentándose con corriente eléctrica, en la fecha y hora que aparece a su izquierda. El código de **evento número 3** indica que el smart meter se desconectó (se le quitó la corriente de alimentación) en la fecha y hora que aparece a su izquierda.

4.3.2. Eventos de detección de fraude

El **nombre lógico** del Profile Generic que registra los eventos de detección de fraude es el **0.99.98.1.255**. Contiene todos los eventos relacionados con la detección de intentos de fraude, tanto por manipulación (por ejemplo, apertura o cierre de la cubierta del meter) como por conexión (por ejemplo, la conexión de un dispositivo externo no autorizado).

Estructura: Fecha y hora | Código de evento

Tamaño buffer: 10 registros.

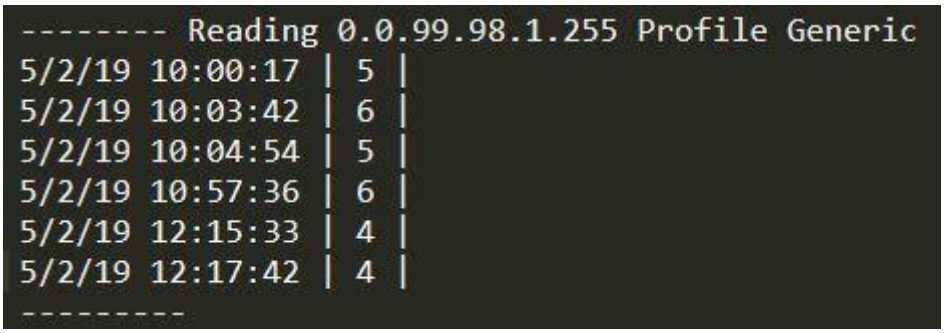

Ilustración 4.5. Eventos del Profile Generic 0.0.99.98.1.255

En la ilustración 4.5. se pueden observar los eventos registrados. El código de **evento número 5** indica que la tapa del smart meter se abrió en la fecha y hora que aparece a su izquierda. El código de **evento número 6** indica que la tapa del smart meter se cerró en la fecha y hora que aparece a su izquierda.

4.3.3. Eventos de control de desconexión

El **nombre lógico** del Profile Generic que registra los eventos de detección de fraude es el **0.99.98.7.255**. Contiene todos los eventos relacionados con todas las comunicaciones que se han realizado con el smart meter a través de vía óptica, PLC o puerto serie.

Estructura: Fecha y hora | Código de evento

Tamaño buffer: 100 registros.

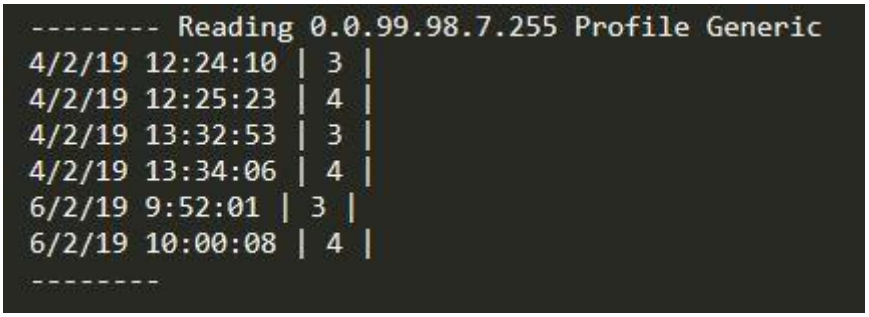

Ilustración 4.6. Eventos del Profile Generic 0.0.99.98.7.255

En la ilustración 4.6. se pueden observar los eventos registrados. El código de **evento número 3** indica que se ha iniciado una conexión con el smart meter a través de su puerto óptico en la fecha y hora que aparece a su izquierda. El código de **evento número 4** indica que se ha cerrado una conexión con el smart meter a través de su puerto óptico en la fecha y hora que aparece a su izquierda.

4.3.4. Evento de sincronización de reloj

El **nombre lógico** del Profile Generic que registra los eventos de detección de fraude es el 0.99.98.8.255. Contiene el registro de todos los cambios de fecha y hora que se han producido en el objeto reloj del smart meter.

Estructura: Fecha y hora antes del cambio | Código de evento | Fecha y hora después del cambio.

Tamaño buffer: 10 registros.

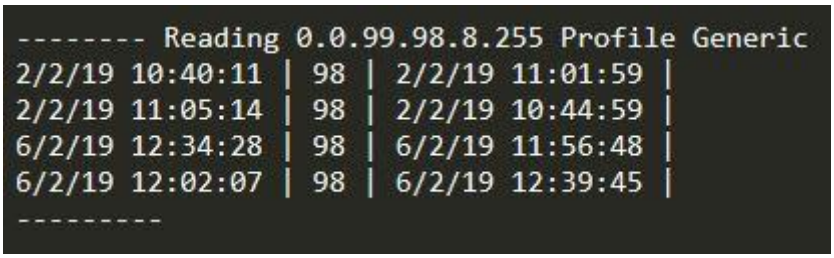

Ilustración 4.7. Eventos del Profile Generic 0.0.99.98.8.255

En la ilustración 4.7. se pueden observar los eventos registrados. El código de **evento número 98** indica que el reloj interno del meter ha sido modificado. La fecha y hora que aparece a la izquierda es la que tenía el smart meter antes de producirse el cambio. Por ende, la fecha y hora que aparece a la derecha es la marca de tiempo que tenía el smart meter después de producirse el cambio.

5. INTERFAZ DE APLICACIÓN

En este epígrafe se desarrolla la explicación del funcionamiento de las interfaces gráficas de la aplicación, en función de cómo interactúe el usuario con la aplicación. Sin embargo, la implementación de los métodos desarrollados para la conexión con el smart meter se recogen en el **epígrafe 3** (Desarrollo de la aplicación).

Como se ha definido en el apartado de objetivo, la funcionalidad de la aplicación Android consiste en establecer conexión entre el smartphone y un smart meter, a través de una sonda óptica que conecta ambos dispositivos. Una vez establecida la conexión, se le permite al usuario realizar dos tipos de lecturas. Por último, una vez que se han leído los datos del smart meter, estos se guardan en un fichero y se suben a un servidor a través de una solicitud HTTP.

La aplicación se desarrolla en dos Activities: Una de inicio de sesión y otra de lectura de datos y envío al servidor.

5.1. Activity de inicio de sesión

La primera **activity** tiene la función de permitir el de **inicio de sesión del smartphone con con el smart meter**, para establecer la relación cliente-servidor.

En esta **activity** se da la opción al usuario de elegir entre **tres los tres tipos de cliente** para conectar con el smart meter, que son: Gestión, Público o Cliente. Además, el usuario debe introducir la contraseña del tipo de cliente que ha seleccionado.

Hay que señalar que **cada tipo de cliente tiene una contraseña única** y es distinta para ambos. Además, ambos parámetros, contraseña y los tres tipos de usuario, **son parámetros internos que posee el smart por defecto**, los cuales no se han modificado.

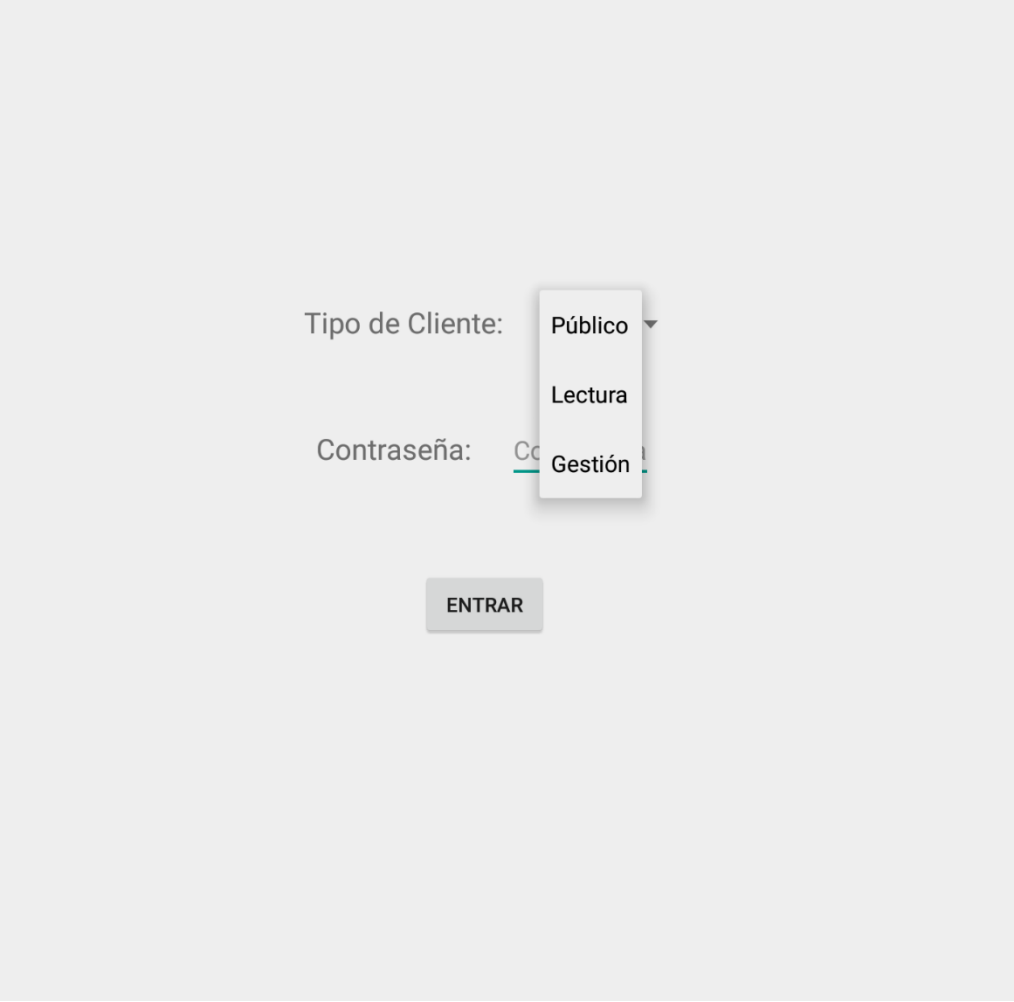

Ilustración 5.1. Interfaz gráfica de la Activity de inicio de sesión

Para facilitar la elección del tipo de cliente al usuario se ha hecho uso de la clase **Spinner** [20] que ofrece Android en su librería por defecto. *En la ilustración 5.1.* se observa el **Spinner** con los tipos de cliente.

Si se produce un error al iniciar sesión se ha implementado un cuadro de diálogo que notifica al usuario el problema ocurrido. Cuando el usuario lo cierra, se le vuelve a dar opción a que vuelva a introducir las credenciales para un nuevo intento de conexión con el smart meter.

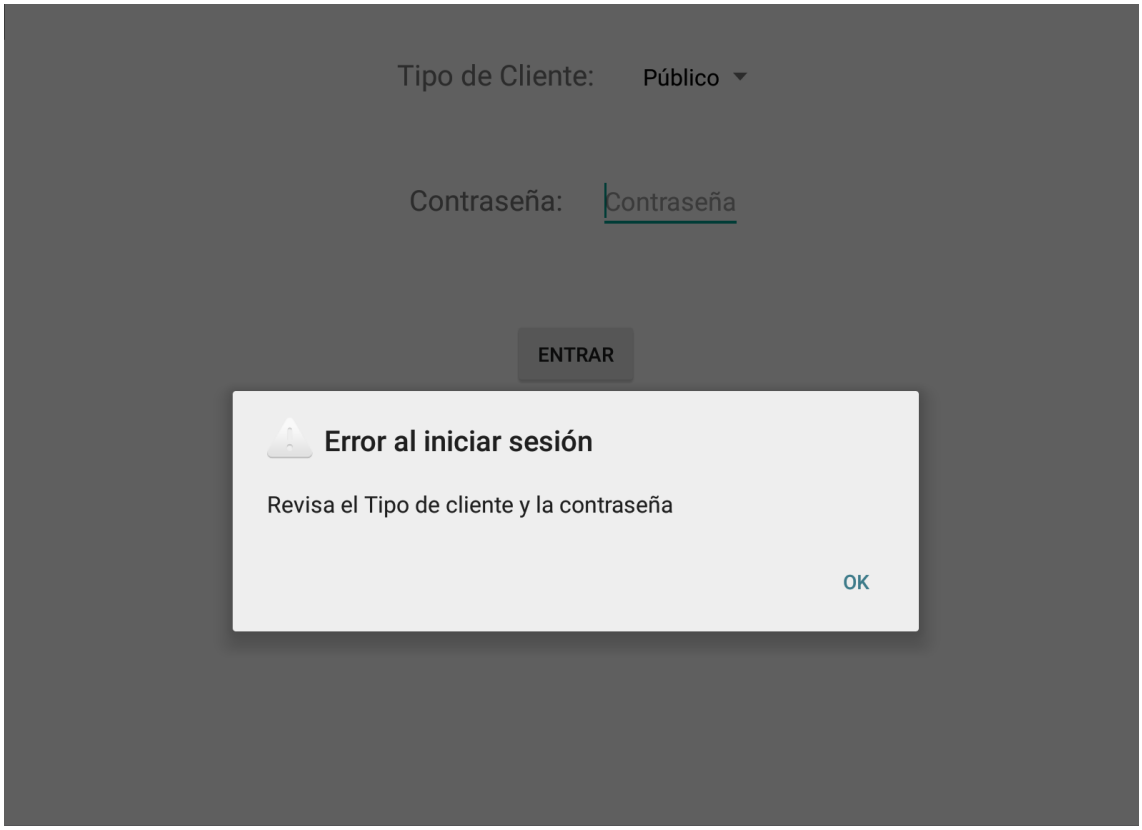

Ilustración 5.2. Diálogo de error de inicio de sesión con el meter

El cuadro de diálogo se ha generado con la clase **AlertDialog** que ofrece Android en su librería por defecto. [21]

Una vez que el usuario introduce correctamente las credenciales y presiona el botón "**entrar**", **se establece la conexión cliente-servidor** entre el smartphone y el smart meter (según se ha explicado en el **epígrafe 3.2**).

Por último, se llama a la función **finish**, la cual implementan todas las activities por defecto (**epígrafe 2.5.1**), para cerrar esta activity y se llama a la **activity de lectura de datos**, pasándole como parámetro la variable de la clase **GXDLMSClient** de **GuruX** con la que se ha establecido la conexión.

5.2. Activity de lectura de datos y envío al servidor

Una vez que se establece la conexión entre el smartphone y el smart meter con éxito, se accede a esta activity.

Ilustración 5.3. Interfaz gráfica de la activity de lectura de datos y envío

Como se puede observar en *la ilustración 5.3.* la interfaz gráfica de la **activity** cuenta con dos botones que permiten al usuario elegir entre la **lectura de la curva de carga** o la **lectura de los eventos registrados** por el smart meter.

Debajo de los botones, aparecen parámetros del smart meter con el que se ha efectuado la conexión, los cuales se ha creído importantes para destacarlos y mostrarlos en la interfaz. Estos parámetros son:

- **Fecha** (Date): muestra la fecha y hora del objeto reloj (nombre lógico **0.0.1.0.0.255**) que posee el smart meter.

- **ID**: Muestra el identificador del smart meter (nombre lógico **0.0.96.1.4.255**). El identificador es único y exclusivo para cada smart meter, viene escrito por defecto en su fabricación y su identificación depende de la compañía que crea el smart meter.

Estos parámetros se han programado para que se lean y se actualicen automáticamente al entrar a esta **activity**. La lectura de estos valores es instantánea y se muestran directamente sin que el usuario tenga que hacer nada. Esto es posible debido a que ya se ha establecido la conexión con el meter en la **activity** de inicio de sesión.

Una vez que el usuario se encuentra en esta **activity** puede escoger entre dos botones: Leer la curva de carga del smart meter o leer todos los eventos registrados. Ambas lecturas envían la información obtenida a un servidor a través de HTTP.

Si el usuario pulsa el botón de **lectura de curva de carga**, la aplicación realiza la lectura del Profile Generic correspondiente, cuyo nombre lógico es **1.0.99.1.0.255**. Los datos obtenidos de esta lectura en este proyecto daban siempre valor nulo debido a que la curva de carga registra el consumo de energía eléctrica del smart meter y con el que se ha trabajado estaba **aislado de la red eléctrica**.

Si el usuario pulsa el botón de lectura de eventos, la aplicación realiza la lectura de todos los Profile Generic correspondientes. Los datos obtenidos de esta lectura con el smart meter con el que se han hecho pruebas se recogen en el **epígrafe 4.3**

5.3. Mejoras implementadas

Ambas lecturas tienen un período de duración de dos minutos aproximadamente. El periodo varía dependiendo de la cantidad de información que tengan los **Profile Generic,** es decir, depende de si el buffer está lleno o no.

Como durante el proceso de lectura, el usuario no interactúa con la aplicación, se ha visto necesario mostrar un cuadro de diálogo que indique al usuario que se está ejecutando la lectura, para que no piense que la aplicación se ha detenido.

En una primera instancia que optó por poner directamente un cuadro de progreso que indicase con un símbolo de cargando, que se estaba ejecutando la lectura. Este cuadro se genera con la clase **ProgressDialog** [22] que ofrece Android en su librería por defecto.

Sin embargo, a la hora de ejecutar la aplicación, el cuadro de progreso no se mostraba durante el proceso de lectura y aunque en la programación se ponía antes del proceso, el cuadro aparecía una vez la lectura había concluido. Esto se debe a que **el proceso de lectura bloqueaba la aplicación** durante el tiempo que se ejecutaba el proceso debido a que el volumen de lectura solicitado al smart meter consumía todos los recursos del smartphone.

Para evitar este problema fue necesario crear un hilo de ejecución secundario dentro del programa, que realice la lectura de los datos requeridos, sin bloquear la aplicación.

5.3.1. Hilos

Todos los componentes de una aplicación Android, tanto las actividades como los servicios se ejecutan en **un mismo hilo de ejecución**, que recibe el nombre de **hilo principal**. Cuando se lanza una nueva aplicación el sistema crea un nuevo hilo principal de ejecución para esta aplicación. Este hilo se encarga de atender los eventos de las distintas funciones. Además, también **es el hilo donde se ejecutan todos los procedimientos que gestionan la interfaz de usuario de la aplicación**.

La solución que se ha escogido para el problema planteado en el **anterior epígrafe** ha consistido en crear un **hilo secundario**, independiente del hilo principal, en el cual se ejecuta la llamada a la función que lee la información del smart meter, que es la que provoca que la interfaz gráfica de la **activity** se quede bloqueada y no muestre el cuadro que notifica el progreso de la lectura.

De esta forma el hilo principal puede seguir atendiendo los eventos requeridos. En concreto la creación de un cuadro de diálogo que muestre al usuario que la aplicación está ejecutando la lectura.

Una vez se llegó a este punto había dos opciones: crear un hilo de forma manual o usar la clase **AsyncTask** que Android proporciona de manera auxiliar. Se optó por la segunda opción [23]

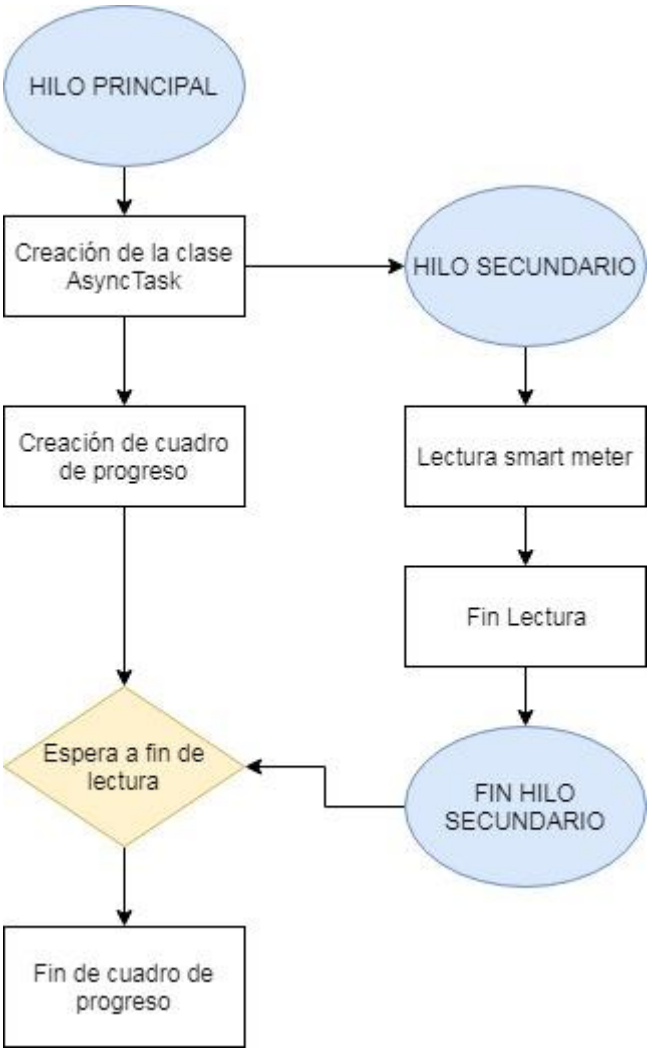

Ilustración 5.4. Diagrama de ejecución de la lectura de datos

En la *ilustración 5.4* se muestra un esquema simplificado de la ejecución del proceso de lectura del smart meter implementando el hilo secundario.

5.3.2. Uso de la clase Asynctask

AsyncTask [23] es una clase auxiliar, implementada por Android, que permite implementar un hilo de ejecución de manera sencilla, mejor organizada y más legible. El procedimiento para implementar **AsyncTask** consiste en crear una nueva clase que extienda de ella y sobrescribir varios de sus métodos entre los que repartiremos la funcionalidad de nuestra tarea. Los métodos a sobreescribir son:

- **OnPreExecute**: Este método sirve para preparar la ejecución de la tarea ya que se ejecuta antes del código principal de la clase, el cual se encuentra en la función **doInBackground**.
  En esta aplicación en concreto se ha utilizado para crear un objeto de la clase **ProgressDialog** para poder notificar al usuario se está realizando la lectura.

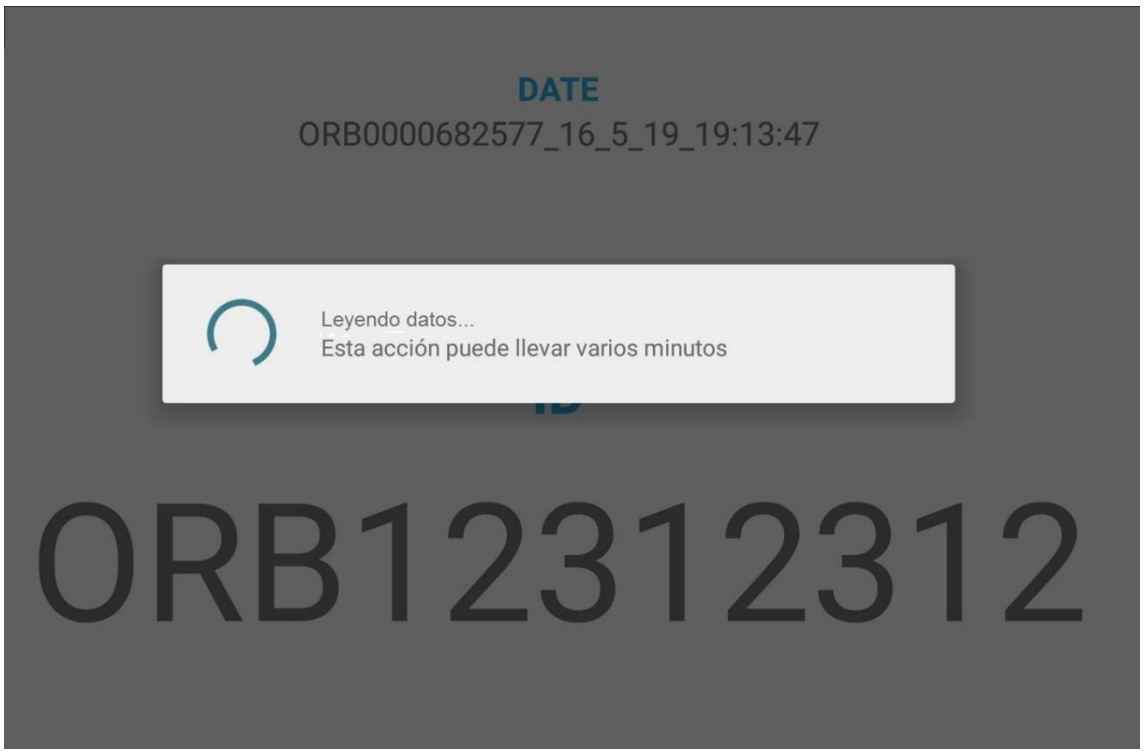

Ilustración 5.5. Cuadro de diálogo generado con la clase ProgressDialog

- **doInBackground**: Este método contiene el código principal de la tarea y es el que se encarga de realizar la tarea en segundo plano.
  En esta aplicación en concreto se ha utilizado para **llamar a la función que realiza la lectura del smart meter**.

- **onProgressUpdate:** Este método permite actualizar la interfaz de la aplicación mientras se ejecuta la tarea en segundo plano. Normalmente se utiliza para mandar mensajes por pantalla al usuario o actualizar una barra para notificarle del progreso de la tarea.
  En esta aplicación en concreto **no se ha utilizado este método**.

- **onPostExecute**: Este método se ejecuta tras terminar **doInBackground** y recibe los parámetros que este le devuelve.
  En esta aplicación en concreto se ha utilizado para cancelar el **ProgressDialog** creado en el método **onPreExecute**, haciendo que desparezca de la pantalla el cuadro de diálogo. Posteriormente, también en este método, se comprueba si la lectura del smart meter se ha completado correctamente. De no ser así, se le indica el error al usuario por pantalla, a través de un **AlertDialog** [21]

De todos estos métodos, solo **doInBackground** se ejecuta en un segundo hilo, mientras que todos los demás se ejecutan en el hilo principal. De esta forma con los métodos restantes se puede hacer referencia al hilo principal, lo que permite actualizar la interfaz de la aplicación mientras se ejecuta el otro hilo en segundo plano.

Para extender la clase **AsyncTask** es necesario especificarle tres tipos de parámetros:

1. El primer tipo de parámetros son los datos que se pasan para iniciar la tarea. Estos parámetros son en forma de array y son los que el método **doInBackground** recibe.
   En esta aplicación en concreto, se pasa el objeto **DLMSClient**, propio de la librería de **GuruX**, con el que se ha **establecido la conexión cliente-servidor entre el smartphone y el smart meter**.

2. El segundo tipo de parámetros se utilizan para actualizar el progreso de la tarea. Estos parámetros son los que recibirá posteriormente el método **onProgressUpdate**.

   En esta aplicación en concreto **no ha sido necesario el uso** de este parámetro, sin embargo, la clase **Asynctask** obliga a escribir uno. Por lo tanto, es necesario escribir la como parámetro la palabra "**Void**"

3. El tercer y último tipo de dato es que devuelve el método **doInBackground**, una vez que se ha finalizado el hilo secundario (cuando finaliza la lectura los datos). Como se ha comentado anteriormente, este parámetro lo recibe el método **onPostExecute**.

   En esta aplicación en concreto el **parámetro** que se pasa es un String **que contiene toda la información que se ha leído del smart meter**.

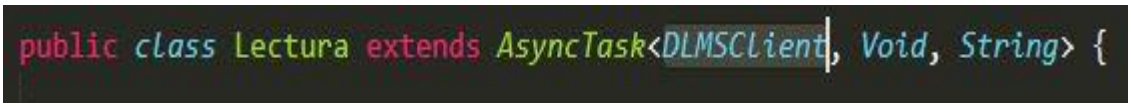

Ilustración 5.6. Extensión de la clase AsyncTask

5.4. Envío de fichero mediante solicitud HTTP

Una vez que se ha leído la información del smart meter la aplicación sube directamente, sin que el usuario tenga que hacer nada, un fichero con los datos leídos a un servidor mediante una petición **HTTP**. El servidor con el que se han hecho las pruebas en un servidor de prueba que permite este tipo de conexiones.

Para realizar una petición **HTTP** es obligatorio crear un hilo secundario de ejecución dentro del programa, debido a que la subida del fichero es un proceso asíncrono, lo que impide su ejecución en el hilo principal. Para ello se volverá a hacer uso de la clase **Asynctask** (*ver epígrafe 5.3.1*).

Al principio de la clase se crea un objeto **FileInputStream** y con él se abre el fichero donde se ha guardado la información leída del smart meter.

A continuación, se crea un objeto de la clase **URL** [25] donde se introduce la dirección del servidor donde se va a subir el archivo. Después se crea un objeto de la clase **HttpURLConnection** [26] dependiente del objeto **URL**, que se va a utilizar para abrir la conexión **HTTP**.

Posteriormente se configura el objeto **HttpURLConnection** y se especifica que la conexión **HTTP** es a través de una solicitud con el método **POST** (*ver epígrafe 2.6*).

Después, se crea una variable **DataOutputStream** y se configura la disposición del fichero a subir junto con el nombre del fichero. El nombre del fichero se compone de la ID del smart meter, más la fecha y hora de su reloj, para posteriormente poder identificar de forma inequívoca el smart meter del cual se ha realizado la lectura.

Por último, se recorren todos los bytes del fichero y se escriben en el **DataOutputStream**,

Finalmente obtenemos la respuesta que nos devuelve el servidor a través del objeto **HttpURLConnection**. Si la respuesta es un 200 significa que la subida se ha realizado con éxito. En ese caso aparece un nuevo cuadro de diálogo, creado con la clase **AlertDialog** [21], que nos indica que se ha realizado la subida correctamente. Al cerrar el cuadro se cierra la aplicación directamente porque se llama al método **finish()**, propio de una **activity** (ver epígrafe 2.5.1).

Ilustración 5.7. Cuadro de diálogo de fin de lectura y envío al servidor

6. CONLUSIONES

Finalmente, tras el estudio del protocolo DLMS/COSEM y haciendo uso del software de GuruX [15] se ha logrado desarrollar con éxito una aplicación para dispositivos Android que permite establecer conexión con un smart meter, leer sus datos y enviarlos dentro de un fichero a un servidor de prueba.

Una vez se cumplió con el objetivo de la aplicación, vi necesario implementar una funcionalidad que notificase al usuario que el proceso de lectura de datos se estaba ejecutando. Sin embargo, una vez implementado, la aplicación se quedaba bloqueada y no mostraba nada por pantalla, tal como se explica en el *epígrafe 5.3*.

Necesité investigar acerca del error puesto que no entendía cuál era el problema, sobre todo buscando casos similares en internet, para darme cuenta de que el proceso de lectura del smart meter bloqueaba la interfaz gráfica de la aplicación. Finalmente averigüe la solución que se detalla en el *epígrafe 5.3*.

Por otra parte, como se ha comentado en anteriores epígrafes de este trabajo, el smart meter con el que se hicieron las pruebas era un smart meter aislado de la red eléctrica. Como futura investigación cabría probar esta aplicación con un smart meter que esté conectado a una red eléctrica y que se encuentre funcionando en su pleno rendimiento.

De esta manera se podrían realizar mejoras en la aplicación y explotar sus recursos, ya que, por ejemplo, la lectura de la curva de carga (que mide el consumo) que implementa la aplicación, siempre daba valores nulos al encontrarse el smart meter aislado de la red eléctrica, lo que resulta obvio.

7. BIBLIOGRAFÍA

[1] S. Galeano. “Hay más móviles conectados a Internet que personas en el mundo”. Marketing4ecommerce. https://marketing4ecommerce.net/moviles-conectados-a-internet/ (acceso: marzo de 2019)

[2] “Diferencias entre Smart Grids y Redes Eléctricas Convencionales”. WordPress. https://globalelectricity.wordpress.com/2013/12/19/diferencias-entre-smart-grids-y-redes-electricas-convencionales/ (acceso: febrero de 2019)

[3] A.J. González López, “Gestión de la energía en una red inteligente”, Trabajo fin de grado, Dpto. de Ingeniería Eléctrica, Universidad Carlos III de Madrid, Madrid, España, 2016. [En línea]. Disponible en: https://e-archivo.uc3m.es/bitstream/handle/10016/14698/PFC_Angel_J_Gonzalez_Lopez.pdf?sequence=1

[4] S. Ledo. “España ya mide la luz con contadores inteligentes”. El Periódico. https://www.elperiodico.com/es/economia/20181021/espana-mide-luz-contadores-inteligentes-7098751 (acceso: febrero de 2019)

[5] Real Decreto 1110/2007, de 24 de agosto, por el que se aprueba el Reglamento unificado de puntos de medida del sistema eléctrico. https://www.boe.es/buscar/doc.php?id=BOE-A-2007-16478

[6] Orden ITC/3860/2007, de 28 de diciembre, por la que se revisan las tarifas eléctricas a partir del 1 de enero de 2008. https://www.boe.es/buscar/doc.php?id=BOE-A-2007-22458

[7] J. Cabeza López-Vázquez, “Estudio de la situación actual de las smart grids”, Trabajo de fin de grado, Dpto. Ingeniería de Telecomunicación, Universidad de Cantabria, Cantabria, España, 2016. [En línea]. Disponible en: https://repositorio.unican.es/xmlui/bitstream/handle/10902/9143/386883.pdf?sequence=1&isAllowed=y

[8] J.A. Vega. “Mooc IV: Respuesta a la demanda t smart grid”. https://blogs.upm.es/itfm/2018/01/14/mooc-iv-respuesta-a-la-demanda-y-smart-grid/ (acceso: marzo de 2019)

[9] “Electricity metering - data exchange for metering reading, tariff and load control - Part 61: Object Identification System (OBIS)” International Electrotechnical Commission, Standard IEC 62056-61, 2002.

[10] Comisión Electrotécnica Internacional (IEC). Disponible en: https://www.iec.ch/ (acceso: marzo de 2019)

[11] RedZ Smart Communications. Disponible en: https://probeformeters.com/es/usb-rs232-and-rs485-probes/kmk116-usb-optical-probe (acceso: enero de 2019)

[12] “Electricity metering - data exchange for metering reading, tariff and load control - Part 21: Direct local data exchange” International Electrotechnical Commission, Standard IEC 62056-61, 2002.

[13] Android Developers. “Android Studio”. Disponible en: https://developer.android.com/studio (acceso: febrero de 2019)

[14] J. Revelo. “Operaciones HTTP En Android Con El Cliente HttpURLConnection”. Hermosa Programación. http://www.hermosaprogramacion.com/2015/01/android-httpurlconnection/ (acceso: abril de 2019)

[15] Gurux Ltd. Disponible en: https://www.gurux.fi/ (acceso: marzo de 2019)

[16] GuruX Ltd. “Gurux DLMS library for Java” Disponible en: https://github.com/Gurux/gurux.dlms.java (acceso: marzo de 2019)

[17] GuruX Ltd. “Gurux Serial for Android” Disponible en: https://github.com/Gurux/gurux.serial.android (acceso: marzo de 2019)

[18] “Yellow Book - DLMS/COSEM Conformance Testing Process” DLMS User Association, Standard IEC 62056-61, Edition 6.1, 2018. Disponible en: https://www.dlms.com/files/Yellow-Book-Ed-61-Excerpt.pdf (acceso: marzo de 2019)

[19] Android Developers. “XMLSerializer”. Disponible en: https://developer.android.com/reference/org/xmlpull/v1/XmlSerializer (acceso: febrero de 2019)

[20] Android Developers. “Controles de números”. Disponible en: https://developer.android.com/guide/topics/ui/controls/spinner (acceso: febrero de 2019)

[21] Android Developers. “Cuadros de diálogo”. Disponible en: https://developer.android.com/guide/topics/ui/dialogs (acceso: febrero de 2019)

[22] Android Developers. “ProgressDialog”. Disponible en: https://developer.android.com/reference/android/app/ProgressDialog (acceso: febrero de 2019)

[23] Android Developers. “AsyncTask”. Disponible en: https://developer.android.com/reference/android/os/AsyncTask (acceso: febrero de 2019)

[24] “Blue Book - COSEM Interface Classes and OBIS Object Identification System”, DLMS User Association, Standard IEC 62056-61, Edition 6.1, 2018. Disponible en: https://www.dlms.com/files/Blue-Book-Ed-122-Excerpt.pdf (acceso: marzo de 2019)

[25] Android Developers. “URL”. Disponible en: https://developer.android.com/reference/java/net/URL (acceso: abril de 2019)

[26] Android Developers. “HttpURLConnection”. Disponible en: https://developer.android.com/reference/java/net/HttpURLConnection (acceso: abril de 2019)

[27] ORBIS. “DOMOTAX TELEGEST PRIME (SMART METER)”. Disponible en: https://www.orbis.es/productos/medida-y-gestion-de-la-energia/contadores-de-telegestion-prime/monofasicos/domotax-telegest-prime-smart-meter (acceso: mayo de 2019)

[28] DLMS User Association. “DLMS/COSEM The standard language for smart devices”. Disponible en: https://www.dlms.com/home (acceso: abril de 2019)